Push-Relabel Algorithms

- Today we will see faster maximum flow algorithms than the $O(|E|^2|V|)$-time Edmonds-Karp algorithm
- We will discuss two so-called "push-relabel" algorithms running in $O(|V|^2|E|)$ and $O(|V|^3)$ time, respectively
- These algorithms iteratively update a "preflow" and a "height function"
  - In each iteration, either a "push" operation is used to update the preflow, or a "relabel" operation is used to update the height function
  - Upon termination, the preflow is a maximum flow

The Concept of a Preflow

- Let $G = (V, E)$ be a given flow network
- A preflow f in G is the same as a flow, except that we relax the flow conservation constraints
  - Let v be a vertex v in $V \setminus \{s, t\}$
  - We require the net flow into v to be nonnegative, rather than requiring it to be zero

Excess Flow and Overflowing Vertices

- Let f be a preflow for a given flow network $G = (V, E)$
- For any vertex v in V, we define the excess flow of v, denoted $e(v)$, as the net flow into v
- For any vertex v in $V \setminus \{s, t\}$, we say that v is "overflowing" if $e(v) > 0$

The Residual Network with Respect to a Preflow

- Let f be a preflow for a given flow network $G = (V, E)$
- Recall our convention that if (u, v) belongs to E, then (v, u) belongs to E
- We define the residual capacity $c_f(u, v)$ of each edge (u, v) as $c(u, v) - f(u, v) + f(v, u)$, as is in the case where f is a flow
  - Note that $c_f(u, v) \geq 0$ since $c(u, v) - f(u, v) \geq 0$ and $f(v, u) \geq 0$
- Likewise, the residual network G_f with respect to a preflow f is defined in the same way as for a flow

The Concept of a Height Function

- Let f be a preflow for a given flow network $G = (V, E)$
- A height function h maps each vertex in V to a nonnegative integer
  - We require that $h(s) = |V|$ and $h(t) = 0$
  - We require that $h(u) \leq h(v) + 1$ for each edge (u, v) with positive residual capacity

A "Configuration"

- Let $G = (V, E)$ be a flow network
- Let f be a preflow for G
- Let h be a height function for G with respect to f
- We call such a pair (f, h) a "configuration"
- The iterative algorithms that we present will maintain a configuration

The Sink is not Reachable in G_f

- Let (f, h) be a configuration for a given flow network $G = (V, E)$
- Lemma 1: There is no path from s to t consisting of edges with positive residual capacity
  - Suppose such a path P exists
  - We can assume without loss of generality that P is simple, so its length is at most $|V| - 1$
  - Since $h(u) \leq h(v) + 1$ for each edge (u, v) on P, we deduce that $h(s) \leq h(t) + |V| - 1$
  - This is a contradiction, since $h(s) = |V|$ and $h(t) = 0$

High-Level Plan

- Let $G = (V, E)$ be a given flow network
- We will iteratively update a configuration for G until we arrive at a configuration (f, h) such that no vertex is overflowing
  - Thus f is a flow
  - By Lemma 1, there is no augmenting path in G_f
  - Since f is a flow and there is no augmenting path in G_f, we conclude that f is a maximum flow for G

The Initial Configuration

- Let $G = (V, E)$ be a given flow network
- We obtain an initial configuration (f, h) for G as follows
  - For each edge (u, v) in E, we set $f(u, v)$ to $c(u, v)$ if $u = s$, and to 0 otherwise
  - For each vertex v in V, we set $h(v)$ to $|V|$ if $v = s$, and to 0 otherwise
  - It is straightforward to verify that (f, h) is a configuration

The Push Operation

- Let (f, h) be a configuration for a given flow network $G = (V, E)$
- A push operation can be applied to an edge (u, v) in E if u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$
- The effect of such a push is to update the preflow f to a new preflow f' such that $f'(u, v) - f'(v, u)$ exceeds $f(u, v) - f(v, u)$ by $\Delta = \min(c_f(u, v), e(u))$
  - The preflow is not modified along edges other than (u, v) and (v, u)
  - If $\Delta = c_f(u, v)$ then $c_{f'}(u, v) = 0$ and we say that the push is "saturating"; otherwise, $\Delta = e(u)$ and the push is "nonsaturating"
  - Observe that (f', h) is a configuration

The Relabel Operation

- Let (f, h) be a configuration for a given flow network $G = (V, E)$, and let u be an overflowing vertex
- Let $E' = \{(u, v) \in E \mid c_f(u, v) > 0\}$
  - Note that E' is nonempty since u is overflowing
- A relabel operation can be applied to u if $h(u) \leq h(v)$ for all edges (u, v) in E'
  - Equivalently, a relabel operation can be applied to u if no push operation can be applied to any edge outgoing from u
- The effect of such a relabel is to increase $h(u)$ to $1 + \min_{(u,v) \in E'} h(v)$
- Observe that the revised (f, h) is a configuration

The Generic Push-Relabel Algorithm

- Let $G = (V, E)$ be a given flow network
- Let (f, h) be the initial configuration for G specified earlier
- While some vertex in G is overflowing
  - Let u be an overflowing vertex
  - Update configuration (f, h) by performing an applicable push or relabel operation associated with vertex u
- Return f, a maximum flow

The Excess at the Source and Sink

- Let $G = (V, E)$ be a given flow network
- Since the sink t is never classified as overflowing, the generic push-relabel algorithm never applies a push operation to an edge leaving t
- Consequently, $e(t) \geq 0$ throughout any execution of the generic push-relabel algorithm
- Thus $e(v) \geq 0$ for all v in $V - s$
- Since $\sum_{v \in V} e(v) = 0$, we conclude that $e(s) \leq 0$

A Reachability Lemma

- Let (f, h) be a configuration for a given flow network $G = (V, E)$, and let u^* be an overflowing vertex
- Let U denote the set of all vertices v such that there is a path from u^* to v in G_f where each edge on the path has positive residual capacity
- Lemma 2: The source s belongs to U
  - Suppose not
  - Observe that $\sum_{u \in U} e(u) = \sum_{u \in U} \sum_{v \in V \setminus U} f(v, u) - f(u, v)$
  - Since the LHS is positive, there are vertices u in U and v in $V \setminus U$ such that $f(v, u) > f(u, v)$; hence $c_f(u, v) > 0$, contradicting the definition of U

Bounding the Vertex Heights

- Let $G = (V, E)$ be a given flow network
- Lemma 3: Throughout any execution of the generic push-relabel algorithm on input G, no vertex height exceeds $2|V| - 1$
  - Suppose the claim is violated in some execution, and the first violation occurs as a result of applying a relabel operation to a vertex u
  - After this relabel operation, vertex u is overflowing and (f, h) is a configuration, so Lemma 2 and the definition of a height function imply that $h(u) \leq h(s) + |V| - 1 = 2|V| - 1$, a contradiction

Bounding the Total Number of Relabel Operations

- Let $G = (V, E)$ be a given flow network
- Lemma 4: The total number of relabel operations performed on a given vertex u in any execution of the generic push-relabel algorithm on G is $O(|V|)$
  - The height $h(u)$ of vertex u is nonnegative
  - Each relabel operation on vertex u increases $h(u)$ by at least 1
  - No other operation affects $h(u)$
- Thus the total number of relabel operations is $O(|V|^2)$

Bounding the Number of Saturating Pushes

- Let $G = (V, E)$ be a given flow network, and let (u, v) be an edge in E
- Suppose that saturating pushes are performed on edge (u, v) at iterations i and i' where $i < i'$; hence a push is performed on edge (v, u) at some iteration i'' such that $i < i'' < i'$
- Let h (resp., h') denote the height function at the start of iteration i (resp., i')
- It is not hard to argue that $h'(u) \geq h(u) + 2$
- Using Lemma 3, we deduce that the number of saturating pushes on edge (u, v) is $O(|V|)$
- Lemma 5: The total number of saturating push operations performed in any execution of the generic push-relabel algorithm on input G is $O(|E| \cdot |V|)$

A Potential Function

- Let (f, h) be a configuration for a given flow network $G = (V, E)$
- We will use a potential function argument to bound the number of nonsaturating push operations
- We define the potential Φ of configuration (f, h) as $\sum_{v:e(v)>0} h(v)$
  - Since $e(s) \leq 0$ and $h(t) = 0$, this is equivalent to the sum of the heights of the overflowing vertices
- The potential Φ is nonnegative
- The total increase in Φ across all relabel operations is $O(|V|^2)$

Bounding the Number of Nonsaturating Pushes

- Each saturating push increases the potential by at most $2|V| - 1 = O(|V|)$
- Each nonsaturating push decreases the potential by at least 1
- Thus the number of nonsaturating push operations is $O(|V|^2 + |V|^2|E|) = O(|V|^2|E|)$

Bounding the Running Time

- Let $G = (V, E)$ be a given flow network
- We have argued that any execution of the generic push-relabel algorithm on G uses $O(|V|^2)$ relabel operations, $O(|E| \cdot |V|)$ saturating push operations, and $O(|V|^2|E|)$ nonsaturating push operations
- It is easy to give an $O(|V|^2|E|)$-time implementation of the generic push-relabel algorithm
  - No special data structures are required, just lists
  - We omit the details, since the next algorithm that we present gives a better time bound

Improving the Running Time

- At any given point in the the execution of the generic push-relabel algorithm, more than one push or relabel operation may be applicable
- By selecting an appropriate operation to perform, we can improve the worst-case running time
- We will sketch an approach that leads to an improved $O(|V|^3)$ bound on the number of nonsaturating push operations
  - Using elementary data structures, we will obtain an $O(|V|^3)$-time implementation

The Admissible Network

- Let (f, h) be a configuration for a given flow network $G = (V, E)$
- We say that an edge (u, v) in E is admissible if $c_f(u, v) > 0$ and $h(u) = h(v) + 1$
  - Note: We do not require u to be overflowing
- We define the admissible network $G_{f,h}$ as $(V, E_{f,h})$ where $E_{f,h}$ denotes the set of admissible edges
- Lemma 6: The admissible network is a DAG
  - This is immediate, since vertex heights decrease along any path

The Effect of a Push on the Admissible Network

- Suppose edge (u, v) is admissible, and vertex u is overflowing
- Thus a push operation can be applied to the edge (u, v)
- What happens to the admissible network if we perform this push operation?
  - No edge is added to the admissible network
  - If the push is saturating, then the edge (u, v) is removed from the admissible network

The Effect of a Relabel on the Admissible Network

- Suppose vertex u is overflowing and has outdegree zero in the admissible network
- Thus a relabel operation can be applied to vertex u
- What happens to the admissible network if we perform this relabel operation?
  - Only edges incident on u are impacted
  - Any edges entering u are removed from the admissible network
  - At least one edge leaving u is added to the admissible network

Adjacency Lists

- Assume that the input flow network $G = (V, E)$ is given in adjacency list format
- For each vertex v in V, we maintain a "current vertex" in the adjacency list of vertex v
  - Initially, the current vertex is the first vertex on the adjacency list of v
  - If the current vertex is nil, it means that the end of the list has been reached

The Discharge Operation Applied to a Vertex u

- While u is overflowing
  - Set v to the current vertex in the adjacency list of u
  - If v is nil, apply a relabel operation to u and set the current vertex of u to the first vertex on the adjacency list of u
  - Otherwise
    - If $c_f(u, v) > 0$ and $h(u) = h(v) + 1$, apply a push operation to edge (u, v)
    - Otherwise, advance the current vertex of u to the next vertex on the adjacency list of u

Correctness of the Discharge Operation

- Lemma 7: Whenever a discharge performs a push operation on an edge (u, v), the push is applicable
  - Immediate
- Lemma 8: Whenever a discharge on vertex u performs a relabel operation, the relabel is applicable
  - The key is to show that we maintain the following invariant: For every vertex v' that precedes the current vertex v on the adjacency list of u, the edge (u, v') does not belong to the admissible network
  - A push operation does not create any admissible edges, and hence cannot produce a violation of the invariant
  - A relabel of a vertex different from u cannot create an admissible edge outgoing from u, and hence cannot produce a violation of the invariant

The Relabel-to-Front Algorithm

- A linked list L of all the vertices in $V \setminus \{s, t\}$ is maintained; the initial order is arbitrary
- A vertex u in L is maintained; initially u is the first vertex in L
- While u is not nil
  - Apply a discharge operation to u
  - If this discharge performed one or more relabel operations on u, then move u to the front of list L
  - Set u to the successor of u on L
- Remark: The relabel-to-front algorithm is a specialization of the generic push-relabel algorithm, so (f, h) is a configuration throughout the execution

Invariants Maintained by the Relabel-to-Front Algorithm

- The algorithm maintains two key invariants
  1. The order of the vertices on L is a topological ordering of the vertices of the admissible network (with s and t removed)
  2. No vertex preceding u on L is overflowing
- The first invariant holds initially because there are no edges in the admissible network
- The second invariant holds initially because u is the first vertex on L

The First Invariant is Maintained

- Push operations cannot falsify the invariant because such operations do not add any edges to the admissible network
- If one or more relabel operations are applied to u during an iteration, then the following conditions hold after the iteration
    - There are no incoming edges to u in the admissible network
    - Any new edges in the admissible network are outgoing from u, which is okay since u gets moved to the front of L

The Second Invariant is Maintained

- If one or more relabel operations are applied to u during an iteration, then the claim is immediate since u gets moved to the front of L
- Otherwise
  - Since push operations do not create admissible edges, the discharge only performed push operations on edges that were admissible at the start of the iteration
  - Since the first invariant held at the start of the iteration, no push is performed (from u) to a vertex that precedes u on L during the iteration
  - Since the second invariant held at the start of the iteration, we deduce that the second invariant holds at the end of the iteration

Partial Correctness of the Relabel-to-Front Algorithm

- Assume an execution of the relabel-to-front algorithm terminates
- The second invariant implies that no vertex is overflowing
  - Remark: When u is nil, then u has reached the end of the list, and so all vertices on L precede u
- Thus f is a flow
- As in the case of the generic push-relabel algorithm, since f is a flow and (f, h) is a configuration, we conclude that f is a maximum flow

Bounding the Running Time

- Let us define a phase as the interval between two successive relabel operations
  - Thus the number of phases is $O(|V|^2)$
  - Each phase performs at most $|V|$ calls to discharge
  - Thus the number of calls to discharge is $O(|V|^3)$
- It is easy to see that the total time spent outside of the calls to discharge is $O(|V|^3)$
- It remains to bound the total time spent within the $O(|V|^3)$ calls to discharge

Analyzing the Total Time Spent within Discharge

- Each iteration of the discharge loop falls into exactly one of three categories
  1. A relabel is performed
  2. A push is performed
  3. The current vertex on the adjacency list of u is advanced
- In what follows, we prove that the total running time for the iterations falling into each of the three categories is $O(|V|^3)$
- For any vertex u in $V \setminus \{s, t\}$, let d_u denote the degree of u in flow network $G = (V, E)$

The Total Time for the First Category (Relabel)

- We can implement a relabel of u in $O(d_u)$ time
- The number of relabel operations of u is $O(|V|)$
- The total running time for the first category is

$$\sum_{u \in V \setminus \{s,t\}} O(d_u|V|) = O(|E| \cdot |V|)$$

The Total Time for the Second Category (Push)

- Each push operation takes $O(1)$ time
- From our analysis of the generic push-relabel algorithm, the total number of saturating push operations is $O(|E| \cdot |V|)$
- There is at most one nonsaturating push per discharge, so the total number of nonsaturating push operations is $O(|V|^3)$
- Thus the total running time for the second category is $O(|V|^3)$

The Total Time for the Third Category (Advance)

- Advancing the current vertex on the adjacency list of u takes $O(1)$ time
- This is done at most d_u times between relabels of u
- Vertex u is relabeled $O(|V|)$ times
- The total running time for the third category is

$$\sum_{u \in V \setminus \{s,t\}} O(d_u |V|) = O(|E| \cdot |V|)$$